UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/790,302	03/01/2004	Michael David Marr	MSFT-3031/306162.01	9280

41505 7590 08/13/2008

WOODCOCK WASHBURN LLP (MICROSOFT CORPORATION)
CIRA CENTRE, 12TH FLOOR
2929 ARCH STREET
PHILADELPHIA, PA 19104-2891

EXAMINER
CHEN, QING

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
08/13/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/790,302	MARR ET AL.	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *05 May 2008*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1,2,5-14,16-18 and 21-24* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1,2,5-14,16-18 and 21-24* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on May 5, 2008.

2. **Claims 1, 2, 5-14, 16-18, and 21-24** are pending.

3. **Claims 1, 5, 6, 12, 18, and 21** have been amended.

4. **Claims 3, 4, 15, 19, 20, and 25** have been cancelled.

5. The objection to the specification is withdrawn in view of Applicant's amendments to the specification.

6. The objection to Claim 21 is withdrawn in view of Applicant's amendments to the claim.

Response to Amendment

Claim Objections

7. **Claims 12-14, 16, and 17** are objected to because of the following informalities:

- **Claim 12** recites the limitations "the first software module," "said second software module," and "said third software module." Applicant is advised to change these limitations to read "the first program module," "said second program module," and "said third program module," respectively, for the purpose of providing them with proper explicit antecedent basis.

- **Claims 13, 14, 16, and 17** depend on Claim 12 and, therefore, suffer the same deficiency as Claim 12.

Appropriate correction is required.

Claim Rejections - 35 USC § 103

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

9. **Claims 1, 2, 5-8, 10, and 11** are rejected under 35 U.S.C. 103(a) as being unpatentable over **US 6,003,095 (hereinafter "Pekowski02")** in view of **US 5,946,486 (hereinafter "Pekowski01")** and **US 6,226,618 (hereinafter "Downs")**.

As per **Claim 1**, Pekowski02 discloses:

- from the first software module, issuing a call to a first method that invokes a functionality performed by the second software module *(see Figure 5: 310 and 330; Column 6: 30-32, "In step 310, a calling executable from the application program calls a particular entry point of the DLL to export a corresponding function from the DLL.")*;

- using a third software module for performing said first method *(see Figure 5: 320; Column 6: 32-34, "The call is then routed to the demand load library code of the demand load library in step 320.")*; and

- returning from said second software module to said first software module that issued the call and bypassing said third software module *(see Figure 5; Column 6: 44-47, "The function to be exported is executed to generate a result in step 330, and the result from step 330 is returned directly to the application program from step 310."; Column 7: 19-24, "The efficiency*

comes from skipping the demand load code on the return path of the call. Because the call appears to the target DLL as if it came from the original DLL, the return from the call goes directly to the original caller's code, skipping the demand load code in between.").

However, Pekowski02 does not disclose:

- one or more stubs that comprise code segments that are callable by the first software module as an intermediary, the one or more stubs for performing said first method, the one or more stubs being used to enter the second software module and identify said functionality, said call being made according to a first calling convention, said third software module using a second calling convention different from said first calling convention to invoke said functionality in the second software module, the second calling convention comprising a non-standard calling convention that preserves a return address across more than one call boundary;

- verifying that the call originated from a source that is permitted to invoke said functionality, the one or more stubs from the third software module comprising data required during said verification by the second software module, said data required during said verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification; and

- performing said functionality that includes sensitive functions at the second software module, the sensitive functions comprising verifying and authenticating the call from the first software module.

Pekowski01 discloses:

- one or more stubs that comprise code segments that are callable by the first software module as an intermediary, the one or more stubs for performing said first method, the one or more stubs being used to enter the second software module and identify said functionality, said call being made according to a first calling convention, said third software module using a second calling convention different from said first calling convention to invoke said functionality in the second software module, the second calling convention comprising a non-standard calling convention that preserves a return address across more than one call boundary *(see Figure 6; Column 9: 43-47, "At the shadow DLL's common entry function, the stack contains the caller's return address 600, the caller's parameters 610, the first time called flag 602, entry only flag 604, hook value 606, and target DLL's entry point address 608." and 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return address 600 and the hook value 606 for entry point allocated by the common entry function to save caller's return address."; Column 10: 25-27, "Call 705 to the entry point to target DLL 710 calls corresponding entry point function 720 in shadow DLL 725." and 31-35, "Shadow DLL 725 contains entry point functions 735, 740, 745 for each entry point in the target DLL 710. After jumping to the common entry code 730, the common entry then jumps to target DLL 710 which then returns to exit function 750 in shadow DLL 725.")*; and

- verifying that the call originated from a source that is permitted to invoke said functionality, the one or more stubs from the third software module comprising data required during said verification by the second software module, said data required during said

verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification *(see Column 9: 40-43, "At the shadow DLL's entry point function, the stack contains the caller's return address 600 and the caller's parameter 610 pushed onto the stack by the caller (see FIG. 5A)."; Column 10: 27-32, "Entry point 720 contains programming code which contains the instruction of jumping to common entry code 730 while passing the parameters of entry point address, hook value, entry only flag, and first time flag. Shadow DLL 725 contains entry point functions 735, 740, 745 for each entry point in the target DLL 710.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include one or more stubs that comprise code segments that are callable by the first software module as an intermediary, the one or more stubs for performing said first method, the one or more stubs being used to enter the second software module and identify said functionality, said call being made according to a first calling convention, said third software module using a second calling convention different from said first calling convention to invoke said functionality in the second software module, the second calling convention comprising a non-standard calling convention that preserves a return address across more than one call boundary; and verifying that the call originated from a source that is permitted to invoke said functionality, the one or more stubs from the third software module comprising data required during said verification by the second software module, said data required during said verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification. The modification would be obvious

because one of ordinary skill in the art would be motivated to trace events occurring upon entries to the target DLL *(see Pekowski01 – Column 5: 3-5)*.

Downs discloses:

- performing said functionality that includes sensitive functions at the second software module, the sensitive functions comprising verifying and authenticating the call from the first software module *(see Column 10: 53-60, "When the Clearinghouse(s) 105 receives a request for a decryption key for the Content 113 from an intermediate or End-User(s), the Clearinghouse(s) 105 validates the integrity and authenticity of the information in the request; verifies that the request was authorized by an Electronic Digital Content Store(s) or Content Provider(s) 101; and verifies that the requested usage complies with the content Usage Conditions as defined by the Content Provider(s) 101.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Downs into the teaching of Pekowski02 to include performing said functionality that includes sensitive functions at the second software module, the sensitive functions comprising verifying and authenticating the call from the first software module. The modification would be obvious because one of ordinary skill in the art would be motivated to validate the integrity and authenticity of the calling executable *(see Downs – Column 10: 53-60)*.

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Pekowski02 further discloses:

- wherein said first method is performed by the second software module, said first method being exposed to the first software module, said first method performing said functionality *(see Figure 5; Column 6: 30-32, "In step 310, a calling executable from the application program calls a particular entry point of the DLL to export a corresponding function from the DLL.")*.

As per **Claim 5**, the rejection of **Claim 1** is incorporated; however, Pekowski02 and Downs do not disclose:

- wherein said second calling convention causes a program stack to be modified in order to cause a next occurring return instruction to cause a return to the location in which a return would have occurred if a return had been executed following a call to said third software module and prior to a call to said second software module.

Pekowski01 discloses:

- wherein said second calling convention causes a program stack to be modified in order to cause a next occurring return instruction to cause a return to the location in which a return would have occurred if a return had been executed following a call to said third software module and prior to a call to said second software module *(see Column 9: 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return address 600 and the hook value 606 for entry point allocated by the common entry function to save caller's return address.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include wherein said second calling convention causes a program stack to be modified in order to cause a next occurring return instruction to cause a return to the location in which a return would have occurred if a return had been executed following a call to said third software module and prior to a call to said second software module. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing.

As per **Claim 6**, the rejection of **Claim 5** is incorporated; however, Pekowski02 and Downs do not disclose:

- wherein said first calling convention comprises placing a first return address on a stack, said first return address representing a location at which execution is to resume after a function call is completed, and wherein said second calling convention comprises placing a second return address on a stack and placing data at the location represented by said second return address, and wherein the method further comprises:
- using said second return address to find said data, said data being used for at least one of:
    - performing said verifying act; and
    - identifying a location to execute in order to perform said functionality.

Pekowski01 discloses:

- wherein said first calling convention comprises placing a first return address on a stack, said first return address representing a location at which execution is to resume after a function call is completed, and wherein said second calling convention comprises placing a second return address on a stack and placing data at the location represented by said second return address *(see Column 9: 43-47, "At the shadow DLL's common entry function, the stack contains the caller's return address 600, the caller's parameters 610, the first time called flag 602, entry only flag 604, hook value 606, and target DLL's entry point address 608." and 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return address 600 and the hook value 606 for entry point allocated by the common entry function to save caller's return address.")*, and wherein the method further comprises:

- using said second return address to find said data, said data being used for at least one of:

    - performing said verifying act; and

    - identifying a location to execute in order to perform said functionality *(see Column 9: 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return address 600 and the hook value 606 for entry point allocated by the common entry function to save caller's return address.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include wherein said first calling convention comprises placing a first return address on a stack, said first return address representing a location at which execution is to resume after a function call is completed, and wherein said second calling convention comprises placing a second return address on a stack and placing data at the location represented by said second return address, and wherein the method further comprises: using said second return address to find said data, said data being used for at least one of: performing said verifying act; and identifying a location to execute in order to perform said functionality. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing.

As per **Claim 7**, the rejection of **Claim 1** is incorporated; however, Pekowski02 and Downs do not disclose:

- examining a call stack to identify a return address, and determining that the return address is part of a program module that is permitted, according to a standard or rule, to invoke said functionality.

Pekowski01 discloses:

- examining a call stack to identify a return address, and determining that the return address is part of a program module that is permitted, according to a standard or rule, to invoke said functionality *(see Column 9: 40-43, "At the shadow DLL's entry point function, the stack*

contains the caller's return address 600 and the caller's parameter 610 pushed onto the stack by the caller (see FIG. 5A).").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include examining a call stack to identify a return address, and determining that the return address is part of a program module that is permitted, according to a standard or rule, to invoke said functionality. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing.

As per **Claim 8**, the rejection of **Claim 7** is incorporated; however, Pekowski02 and Downs do not disclose:

- determining that said return address is from a location or range of locations within said first software module from which invocation of said functionality is permitted to originate.

Pekowski01 discloses:

- determining that said return address is from a location or range of locations within said first software module from which invocation of said functionality is permitted to originate *(see Column 9: 40-43, "At the shadow DLL's entry point function, the stack contains the caller's return address 600 and the caller's parameter 610 pushed onto the stack by the caller (see FIG. 5A).").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02

to include determining that said return address is from a location or range of locations within said first software module from which invocation of said functionality is permitted to originate. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing.

As per **Claim 10**, the rejection of **Claim 1** is incorporated; and Pekowski02 further discloses:

- wherein said first software module is, or is part of, an application program *(see Column 6: 30-32, "In step 310, a calling executable from the application program calls a particular entry point of the DLL to export a corresponding function from the DLL.").*

As per **Claim 11**, the rejection of **Claim 1** is incorporated; and Pekowski02 further discloses:

- wherein said second software module comprises a dynamic-link library *(see Column 6: 30-32, "In step 310, a calling executable from the application program calls a particular entry point of the DLL to export a corresponding function from the DLL.").*

10. **Claim 9** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Pekowski02** in view of **Pekowski01** and **Downs** as applied to Claim 1 above, and further in view of **US 6,880,149 (hereinafter "Cronce")**.

As per **Claim 9**, the rejection of **Claim 1** is incorporated; however, Pekowski02, Pekowski01, and Downs do not disclose:

- verifying that said first software module, or a portion thereof, has not been modified relative to a previously-known state.

Cronce discloses:

- verifying that said first software module, or a portion thereof, has not been modified relative to a previously-known state *(see Column 2: 3-14, "The resulting executable code is delivered as a protected software application that generates a new checksum at runtime and compares it with the computed checksum, and determines that the software program has been modified if the checksums fail to match.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Cronce into the teaching of Pekowski02 to include verifying that said first software module, or a portion thereof, has not been modified relative to a previously-known state. The modification would be obvious because one of ordinary skill in the art would be motivated to validate code base integrity *(see Cronce – Column 2: 15-17)*.

11. **Claims 12-14, 16, and 17** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Pekowski02** in view of **Pekowski01**, **Cronce**, and **Downs**.

As per **Claim 12**, Pekowski02 discloses:

- the first program module being called by the second program module via a third program module *(see Figure 5: 310, 320, and 330; Column 6: 30-34, "In step 310, a calling executable from the application program calls a particular entry point of the DLL to export a corresponding function from the DLL. The call is then routed to the demand load library code of the demand load library in step 320.")*; and

- permitting execution of said first program module to proceed and returning to said second program module which issued the call and bypassing said third program module *(see Figure 5; Column 6: 44-47, "The function to be exported is executed to generate a result in step 330, and the result from step 330 is returned directly to the application program from step 310."; Column 7: 19-24, "The efficiency comes from skipping the demand load code on the return path of the call. Because the call appears to the target DLL as if it came from the original DLL, the return from the call goes directly to the original caller's code, skipping the demand load code in between.")*.

However, Pekowski02 does not disclose:

- examining a call stack of a process in which said first program module executes to identify a return address in which control of the process will return upon completion of a call to said first program module;

- determining that said return address is located within a second program module that is permitted to call said first program module, said determining comprising checking a datum that represents a calling code used by the second program module, the datum being derived from a portion or the entirety of the second program module;

- the third program module having one or more stubs with code segments that are callable by the second program module as an intermediary, the one or more stubs comprising data required during a verification by the first program module, said data required during said verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification; and

- wherein said first program module comprises cryptographic functionality that stores and obscures a decryption key and that uses said decryption key to decrypt content.

Pekowski01 discloses:

- examining a call stack of a process in which said first program module executes to identify a return address in which control of the process will return upon completion of a call to said first program module *(see Column 9: 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return address 600 and the hook value 606 for entry point allocated by the common entry function to save caller's return address.")*;

- determining that said return address is located within a second program module that is permitted to call said first program module *(see Column 9: 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return*

address 600 and the hook value 606 for entry point allocated by the common entry function to save caller's return address."); and

- the third program module having one or more stubs with code segments that are callable by the second program module as an intermediary, the one or more stubs comprising data required during a verification by the first program module, said data required during said verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification *(see Figure 6; Column 10: 25-32, "Call 705 to the entry point to target DLL 710 calls corresponding entry point function 720 in shadow DLL 725." and 31-35, "Shadow DLL 725 contains entry point functions 735, 740, 745 for each entry point in the target DLL 710. After jumping to the common entry code 730, the common entry then jumps to target DLL 710 which then returns to exit function 750 in shadow DLL 725. Entry point 720 contains programming code which contains the instruction of jumping to common entry code 730 while passing the parameters of entry point address, hook value, entry only flag, and first time flag. Shadow DLL 725 contains entry point functions 735, 740, 745 for each entry point in the target DLL 710.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include examining a call stack of a process in which said first program module executes to identify a return address in which control of the process will return upon completion of a call to said first program module; determining that said return address is located within a second program module that is permitted to call said first program module; and the third program

module having one or more stubs with code segments that are callable by the second program module as an intermediary, the one or more stubs comprising data required during a verification by the first program module, said data required during said verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing and trace events occurring upon entries to the target DLL *(see Pekowski01 – Column 5: 3-5)*.

Cronce discloses:

- said determining comprising checking a datum that represents a calling code used by the second program module, the datum being derived from a portion or the entirety of the second program module *(see Column 2: 3-14, "The resulting executable code is delivered as a protected software application that generates a new checksum at runtime and compares it with the computed checksum, and determines that the software program has been modified if the checksums fail to match."; Column 3: 19-21, "If the checksums compare, then the code is not changed, and execution begins of the main application code 100.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Cronce into the teaching of Pekowski02 to include said determining comprising checking a datum that represents a calling code used by the second program module, the datum being derived from a portion or the entirety of the second program module. The modification would be obvious because one of ordinary skill in the art would be motivated to validate code base integrity *(see Cronce – Column 2: 15-17)*.

Downs discloses:

- wherein said first program module comprises cryptographic functionality that stores and obscures a decryption key and that uses said decryption key to decrypt content *(see Column 10: 61-65, "Once these verifications are satisfied, the Clearinghouse(s) 105 sends the decryption key for the Content 113 to the requesting End-User(s) packed in a License SC. The key is encrypted in a manner so that only the authorized user can retrieve it.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Downs into the teaching of Pekowski02 to include wherein said first program module comprises cryptographic functionality that stores and obscures a decryption key and that uses said decryption key to decrypt content. The modification would be obvious because one of ordinary skill in the art would be motivated to allow only the authorized user to have access to the secured content *(see Downs – Column 10: 41-43)*.

As per **Claim 13**, the rejection of **Claim 12** is incorporated; however, Pekowski02, Pekowski01, and Downs do not disclose:

- determining that said second program module, or a portion thereof, has not been modified relative to a previously-known state of said second program module, wherein said act of permitting execution of said first program module to proceed is further based on the determination as to whether said second program module has been modified.

Cronce discloses:

- determining that said second program module, or a portion thereof, has not been modified relative to a previously-known state of said second program module, wherein said act

of permitting execution of said first program module to proceed is further based on the determination as to whether said second program module has been modified *(see Column 2: 3-14, "The resulting executable code is delivered as a protected software application that generates a new checksum at runtime and compares it with the computed checksum, and determines that the software program has been modified if the checksums fail to match."; Column 3: 19-21, "If the checksums compare, then the code is not changed, and execution begins of the main application code 100.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Cronce into the teaching of Pekowski02 to include determining that said second program module, or a portion thereof, has not been modified relative to a previously-known state of said second program module, wherein said act of permitting execution of said first program module to proceed is further based on the determination as to whether said second program module has been modified. The modification would be obvious because one of ordinary skill in the art would be motivated to validate code base integrity *(see Cronce – Column 2: 15-17)*.

As per **Claim 14**, the rejection of **Claim 12** is incorporated; however, Pekowski02, Pekowski01, and Cronce do not disclose:

- wherein said first program module includes logic that resists misuse and/or tampering with said first program module, and wherein said determining act is performed by said first program module.

Downs discloses:

- wherein said first program module includes logic that resists misuse and/or tampering with said first program module, and wherein said determining act is performed by said first program module *(see Column 80: 60-64, "IBM's tamper-resistant software made it difficult to circumvent these copy protection mechanisms. This is a very typical application for tamper-resistant software; the software is used to enforce rules on the usage of some protected type of Content 113.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Downs into the teaching of Pekowski02 to include wherein said first program module includes logic that resists misuse and/or tampering with said first program module, and wherein said determining act is performed by said first program module. The modification would be obvious because one of ordinary skill in the art would be motivated to deter unauthorized entry into a computer software application by a hacker *(see Downs – Column 80: 41-43)*.

As per **Claim 16**, the rejection of **Claim 12** is incorporated; however, Pekowski02, Cronce, and Downs do not disclose:

- wherein said first program module is called by a third program module, said third program module having a callable method exposed to said second program module, said callable method causing said first program module to be invoked and passing said return address to said first program module at the time that said first program module is invoked.

Pekowski01 discloses:

- wherein said first program module is called by a third program module, said third program module having a callable method exposed to said second program module, said callable method causing said first program module to be invoked and passing said return address to said first program module at the time that said first program module is invoked *(see Figure 3; Column 4: 67 to Column 5: 1-3, "... the shadow DLL acts as an interceptor of all calls being made to the target DLL, and thus each call to the target DLL and return from the target DLL passes through the shadow DLL.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include wherein said first program module is called by a third program module, said third program module having a callable method exposed to said second program module, said callable method causing said first program module to be invoked and passing said return address to said first program module at the time that said first program module is invoked. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing.

As per **Claim 17**, the rejection of **Claim 16** is incorporated; however, Pekowski02, Cronce, and Downs do not disclose:

- wherein said first program module adjusts the content of said call stack to reflect that said first program module will return to said return address upon completion of execution of said call to said first program module, said call stack reflecting, in the absence of the adjustment, that said first program module will return to an address other than said return address.

Pekowski01 discloses:

- wherein said first program module adjusts the content of said call stack to reflect that said first program module will return to said return address upon completion of execution of said call to said first program module, said call stack reflecting, in the absence of the adjustment, that said first program module will return to an address other than said return address *(see Column 9: 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return address 600 and the hook value 606 for entry point allocated by the common entry function to save caller's return address.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include wherein said first program module adjusts the content of said call stack to reflect that said first program module will return to said return address upon completion of execution of said call to said first program module, said call stack reflecting, in the absence of the adjustment, that said first program module will return to an address other than said return address. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing.

12. **Claims 18 and 21-23** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Pekowski02** in view of **Pekowski01** and **Cronce**.

As per **Claim 18**, Pekowski02 discloses:

- a function that is performable on behalf of a calling entity *(see Figure 5; Column 6: 30-32, "In step 310, a calling executable from the application program calls a particular entry point of the DLL to export a corresponding function from the DLL.")*; and

- wherein the program module upon completing said function bypasses the intermediate entity and returns to the calling entity's return address *(see Figure 5; Column 6: 44-47, "The function to be exported is executed to generate a result in step 330, and the result from step 330 is returned directly to the application program from step 310."; Column 7: 19-24, "The efficiency comes from skipping the demand load code on the return path of the call. Because the call appears to the target DLL as if it came from the original DLL, the return from the call goes directly to the original caller's code, skipping the demand load code in between.")*.

However, Pekowski02 does not disclose:

- logic that verifies an identity of the calling entity as a condition for performing said function, said logic consulting a call stack in order to identify said calling entity and determining said identity based on a return address on said call stack, said return address representing a location of an instruction to be executed when the program module completes execution, said logic checking a datum that represents a calling code used by the calling entity, the datum being derived from a portion or the entirety of the calling entity; and

- wherein said function is not exposed to said calling entity, and wherein said function is exposed to an intermediate entity that is callable by said calling entity, said intermediate entity calling upon the program module to perform said function on behalf of said calling entity, said

intermediate entity comprising one or more stubs that comprise data required by the logic to verify the identity of the calling entity, the data being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify the function.

Pekowski01 discloses:

- logic that verifies an identity of the calling entity as a condition for performing said function, said logic consulting a call stack in order to identify said calling entity and determining said identity based on a return address on said call stack, said return address representing a location of an instruction to be executed when the program module completes execution *(see Column 9: 40-43, "At the shadow DLL's entry point function, the stack contains the caller's return address 600 and the caller's parameter 610 pushed onto the stack by the caller (see FIG. 5A).")*; and

- wherein said function is not exposed to said calling entity, and wherein said function is exposed to an intermediate entity that is callable by said calling entity, said intermediate entity calling upon the program module to perform said function on behalf of said calling entity, said intermediate entity comprising one or more stubs that comprise data required by the logic to verify the identity of the calling entity, the data being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify the function *(see Figure 6; Column 9: 40-43, "At the shadow DLL's entry point function, the stack contains the caller's return address 600 and the caller's parameter 610 pushed onto the stack by the caller (see FIG. 5A)."; Column 10: 27-32, "Entry point 720 contains programming code which contains the instruction of jumping to common entry code 730 while passing the parameters of*

entry point address, hook value, entry only flag, and first time flag. Shadow DLL 725 contains entry point functions 735, 740, 745 for each entry point in the target DLL 710.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include logic that verifies an identity of the calling entity as a condition for performing said function, said logic consulting a call stack in order to identify said calling entity and determining said identity based on a return address on said call stack, said return address representing a location of an instruction to be executed when the program module completes execution, said logic checking a datum that represents a calling code used by the calling entity, the datum being derived from a portion or the entirety of the calling entity; and wherein said function is not exposed to said calling entity, and wherein said function is exposed to an intermediate entity that is callable by said calling entity, said intermediate entity calling upon the program module to perform said function on behalf of said calling entity, said intermediate entity comprising one or more stubs that comprise data required by the logic to verify the identity of the calling entity, the data being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify the function. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing and trace events occurring upon entries to the target DLL *(see Pekowski01 – Column 5: 3-5).*

Cronce discloses:

- said logic checking a datum that represents a calling code used by the calling entity, the datum being derived from a portion or the entirety of the calling entity *(see Column 2: 3-14,*

"The resulting executable code is delivered as a protected software application that generates a new checksum at runtime and compares it with the computed checksum, and determines that the software program has been modified if the checksums fail to match."; Column 3: 19-21, "If the checksums compare, then the code is not changed, and execution begins of the main application code 100.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Cronce into the teaching of Pekowski02 to include said logic checking a datum that represents a calling code used by the calling entity, the datum being derived from a portion or the entirety of the calling entity. The modification would be obvious because one of ordinary skill in the art would be motivated to validate code base integrity *(see Cronce – Column 2: 15-17).*

As per **Claim 21**, the rejection of **Claim 18** is incorporated; however, Pekowski02 and Cronce do not disclose:

- wherein said calling entity calls said intermediate entity using a first calling convention, and wherein said intermediate entity calls the program module using a second calling convention different from said first calling convention.

Pekowski01 discloses:

- wherein said calling entity calls said intermediate entity using a first calling convention, and wherein said intermediate entity calls the program module using a second calling convention different from said first calling convention *(see Column 9: 43-47, "At the shadow DLL's common entry function, the stack contains the caller's return address 600, the*

caller's parameters 610, the first time called flag 602, entry only flag 604, hook value 606, and target DLL's entry point address 608." and 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return address 600 and the hook value 606 for entry point allocated by the common entry function to save caller's return address.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include wherein said calling entity calls said intermediate entity using a first calling convention, and wherein said intermediate entity calls the program module using a second calling convention different from said first calling convention. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing.

As per **Claim 22**, the rejection of **Claim 21** is incorporated; however, Pekowski02 and Cronce do not disclose:

- wherein said calling entity calls said intermediate entity by calling a function in said intermediate entity and placing a first return address on said call stack, and wherein said intermediate entity calls the program module by calling or jumping to a location in said program module with one or more parameters including said first return address, wherein the program module verifies that said first return address is located within a calling entity that is allowed to

call the program module, and wherein the program module adjusts said call stack so that a return address to be followed upon a next return instruction is equal to said first return address.

Pekowski01 discloses:

- wherein said calling entity calls said intermediate entity by calling a function in said intermediate entity and placing a first return address on said call stack, and wherein said intermediate entity calls the program module by calling or jumping to a location in said program module with one or more parameters including said first return address, wherein the program module verifies that said first return address is located within a calling entity that is allowed to call the program module, and wherein the program module adjusts said call stack so that a return address to be followed upon a next return instruction is equal to said first return address *(see Column 9: 43-47, "At the shadow DLL's common entry function, the stack contains the caller's return address 600, the caller's parameters 610, the first time called flag 602, entry only flag 604, hook value 606, and target DLL's entry point address 608.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include wherein said calling entity calls said intermediate entity by calling a function in said intermediate entity and placing a first return address on said call stack, and wherein said intermediate entity calls the program module by calling or jumping to a location in said program module with one or more parameters including said first return address, wherein the program module verifies that said first return address is located within a calling entity that is allowed to call the program module, and wherein the program module adjusts said call stack so that a return address to be followed upon a next return instruction is equal to said first return address. The

modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing.

As per **Claim 23**, the rejection of **Claim 21** is incorporated; however, Pekowski02 and Cronce do not disclose:

- wherein said first calling convention comprises placing a return address on said call stack, and wherein said second calling convention comprises placing a second return address on said call stack and placing data at the location represented by said second return address, said second return address being used to find said data, said data being used to perform at least one of: verification of the identity of the calling entity; and identification of a location at which said function is located.

Pekowski01 discloses:

- wherein said first calling convention comprises placing a return address on said call stack, and wherein said second calling convention comprises placing a second return address on said call stack and placing data at the location represented by said second return address, said second return address being used to find said data, said data being used to perform at least one of: verification of the identity of the calling entity; and identification of a location at which said function is located *(see Column 9: 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return address 600 and the hook*

value 606 for entry point allocated by the common entry function to save caller's return address.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include wherein said first calling convention comprises placing a return address on said call stack, and wherein said second calling convention comprises placing a second return address on said call stack and placing data at the location represented by said second return address, said second return address being used to find said data, said data being used to perform at least one of: verification of the identity of the calling entity; and identification of a location at which said function is located. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes executing.

13. **Claim 24** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Pekowski02** in view of **Pekowski01**.

As per **Claim 24**, Pekowski02 discloses:

- receiving a first call or first jump from a first entity *(see Figure 5; Column 6: 32-34, "The call is then routed to the demand load library code of the demand load library in step 320.")*;

- issuing a second call or a second jump to a second entity *(see Figure 5; Column 6: 41-44, "After the process exits the demand load initialization routine in step 324, the process*

proceeds to step 326 wherein the program jumps indirectly to the address of the function to be exported."); and

- wherein said second entity returns directly to said first entity without returning to an intermediate entity that received said first call or said first jump *(see Figure 5; Column 6: 44-47, "The function to be exported is executed to generate a result in step 330, and the result from step 330 is returned directly to the application program from step 310."; Column 7: 19-24, "The efficiency comes from skipping the demand load code on the return path of the call. Because the call appears to the target DLL as if it came from the original DLL, the return from the call goes directly to the original caller's code, skipping the demand load code in between.").*

However, Pekowski02 does not disclose:

- there being a call stack which, at the time of said first call or first jump, has a state in which a return address to be executed upon a next return instruction is equal to a first value; and

- the second call or second jump being parameterized by one or more values including said first value, said second entity having access to a second value and using said second value to verify which return address was applicable at the time that said first call or said first jump was made, said second entity adjusting said call stack to set the return address equal to the first value.

Pekowski01 discloses:

- there being a call stack which, at the time of said first call or first jump, has a state in which a return address to be executed upon a next return instruction is equal to a first value *(see Figure 3; Column 4: 67 to Column 5: 1-3, "... the shadow DLL acts as an interceptor of all calls being made to the target DLL, and thus each call to the target DLL and return from the target DLL passes through the shadow DLL."; Column 9: 43-47, "At the shadow DLL's common entry*

function, the stack contains the caller's return address 600, the caller's parameters 610, the first time called flag 602, entry only flag 604, hook value 606, and target DLL's entry point address 608."); and

- the second call or second jump being parameterized by one or more values including said first value, said second entity having access to a second value and using said second value to verify which return address was applicable at the time that said first call or said first jump was made, said second entity adjusting said call stack to set the return address equal to the first value *(see Figure 3; Column 9: 52-59, "At the target DLL's entry point function, the stack contains the caller's parameters 610 pushed onto the stack by the caller and the return address 630 of element allocated from the return function table replaced by the common entry function (see FIG. 5C). The return address table contains the caller's return address 600 and the hook value 606 for entry point allocated by the common entry function to save caller's return address.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Pekowski01 into the teaching of Pekowski02 to include there being a call stack which, at the time of said first call or first jump, has a state in which a return address to be executed upon a next return instruction is equal to a first value; and the second call or second jump being parameterized by one or more values including said first value, said second entity having access to a second value and using said second value to verify which return address was applicable at the time that said first call or said first jump was made, said second entity adjusting said call stack to set the return address equal to the first value. The modification would be obvious because one of ordinary skill in the art would be motivated to keep track of the point to which each active caller should return control when it finishes

executing and trace events occurring upon entries to the target DLL *(see Pekowski01 – Column 5: 3-5)*.

Response to Arguments

14. Applicant's arguments with respect to Claims 1, 12, 18, and 24 have been considered but are moot in view of the new ground(s) of rejection.

In the Remarks, Applicant argues:

a) Another difference between currently amended claim 1 and the cited references is that claim 1 recites "verifying that the call originated from a source that is permitted to invoke said functionality, the one or more stubs from the third software module comprising data required during said verification by the second software module, said data required during said verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification". This is much different than that which is described in Col. 10, lines 27-30 of Pekowski01 which fails to teach including information that is used to identify a function that will be invoked after verification.

Examiner's response:

a) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

Second, Pekowski01 clearly discloses "verifying that the call originated from a source that is permitted to invoke said functionality, the one or more stubs from the third software module comprising data required during said verification by the second software module, said data required during said verification being mixed into instruction streams provided by the one or more stubs, the data also comprising information that is used to identify a function that will be invoked after the verification" *(see Column 9: 40-43, "At the shadow DLL's entry point function, the stack contains the caller's return address 600 and the caller's parameter 610 pushed onto the stack by the caller (see FIG. 5A)."; Column 10: 27-32, "Entry point 720 contains programming code which contains the instruction of jumping to common entry code 730 while passing the parameters of entry point address, hook value, entry only flag, and first time flag. Shadow DLL 725 contains entry point functions 735, 740, 745 for each entry point in the target DLL 710.").* Note that the parameters of entry point address, hook value, entry only flag, and first time flag (information that is used to identify a function) identify the entry point function that will be invoked after the caller's return address is pushed onto the stack (after the verification).

Conclusion

15. Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The

Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/QC/
August 6, 2008

/Wei Zhen/

Supervisory Patent Examiner, Art Unit 2191